SSIS : sample project

Using SSIS as ETL tool

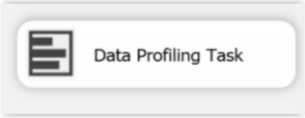
Start Visual Studio 2017 (SSDT) from the Start menu:

Data Exploration

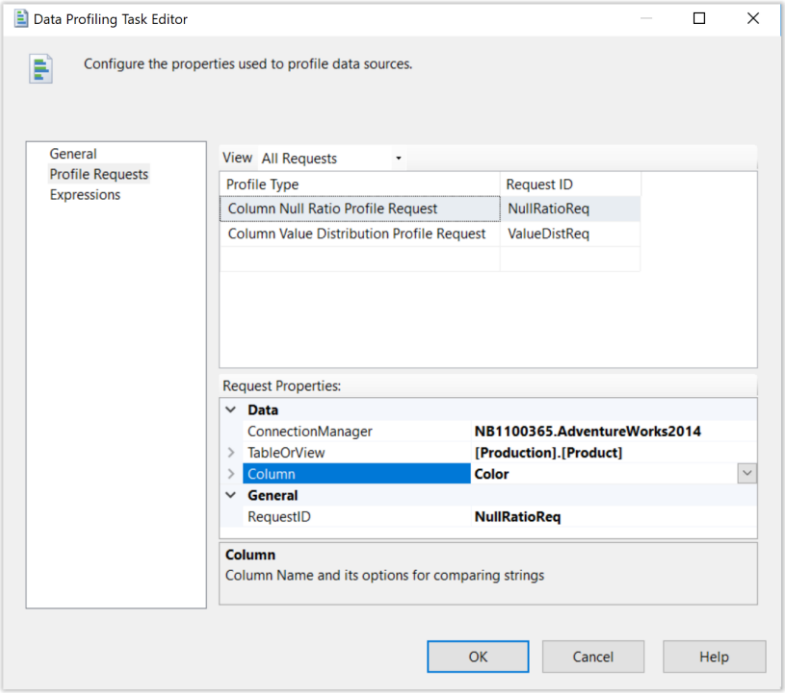
In SSDT create a new Business Intelligence/Integration Services project "AdventureWorksExpl".

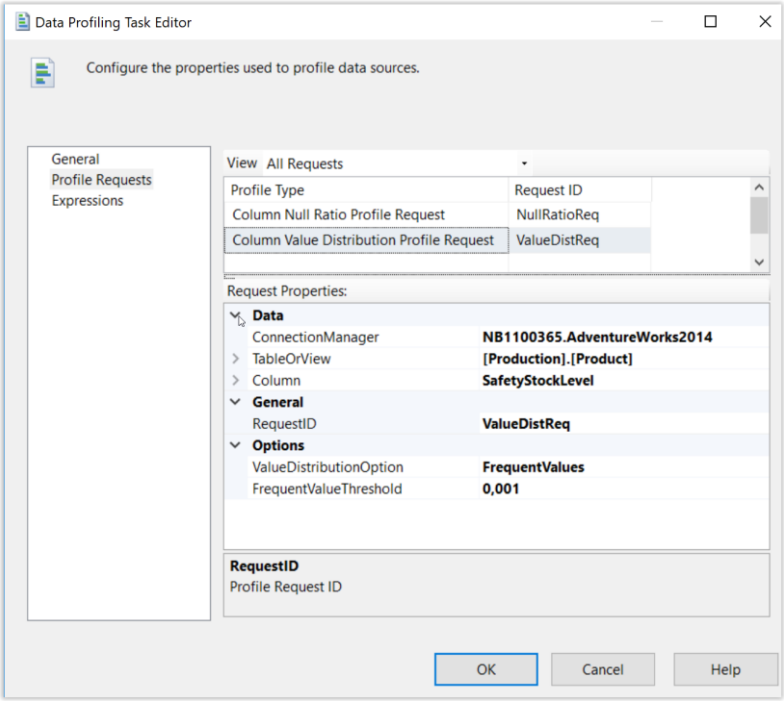
To be able to decide if it's necessary to copy a field to the datawarehouse or to determine the optimal data type and size, you can do some data profiling before starting the ETL process.

Add a Data Profiling Task:

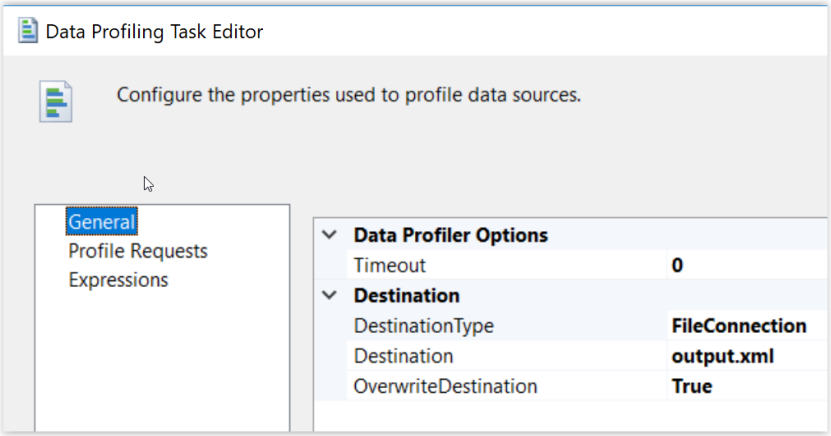


Add two profile requests:

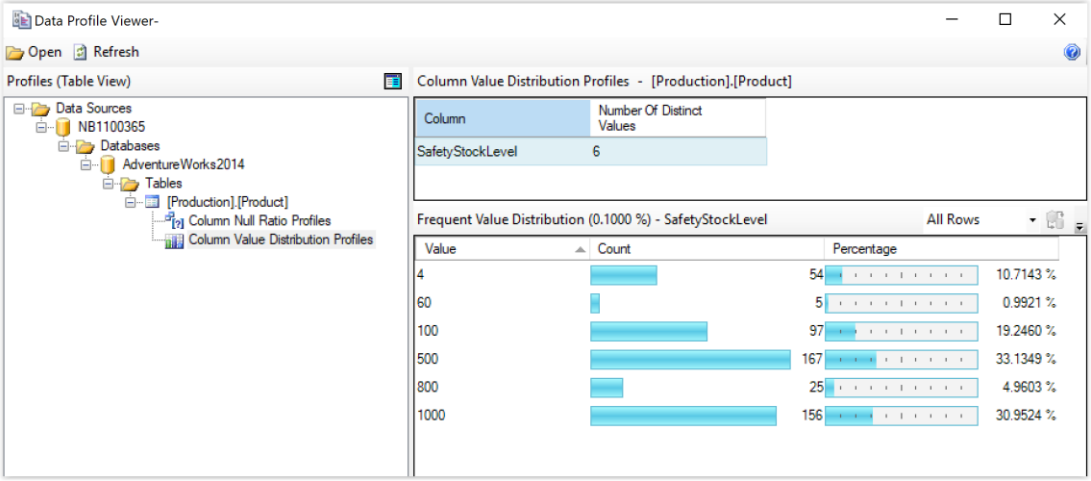




In "General" et destination and select 'Create File'. Enter a **path\output.xml**. OverwriteDestination should be True:



Click OK and click the green arrow to start. After you ran the task, you will see a green V. Double click Data Profile Task and Open Profile Viewer to see the results.



Project setup

In SSDT create a new Business Intelligence/Integration Services project "AdventureWorks".

In SQL Server Management Studio create a database "AdventureWorksDW".

Create in that database following tables:

DimSalesTerritory

Column Name	Data Type	Allow Nu
SalesTerritoryKey	int	☐
SalesTerritoryAlternateKey	int	☑
SalesTerritoryRegion	nvarchar(50)	☐
SalesTerritoryCountry	nvarchar(50)	☐
SalesTerritoryGroup	nvarchar(50)	☑

DimDate:

Column Name	Data Type	Allow Nulls
DateKey	int	☐
FullDateAlternateKey	date	☐
EnglishDayNameOfWeek	varchar(50)	☐
DutchDayNameOfWeek	varchar(50)	☐
MonthNumber	tinyint	☐
EnglishMonthName	varchar(50)	☐
DutchMonthName	varchar(50)	☐
CalenderQuarter	tinyint	☐
CalenderYear	smallint	☐

Prepare ETL process

Create two views in the operational database:

```
create view VwSalesTerritory
as
select t.TerritoryID SalesTerritoryKey,t.TerritoryID SalesTerritoryAlternateKey,
t.Name SalesTerritoryRegion,c.Name SalesTerritoryCountry,t.[Group] SalesTerritoryGroup
from sales.SalesTerritory t join person.CountryRegion c
on t.CountryRegionCode=c.CountryRegionCode;

create view VwDimDate as
select distinct cast(format(OrderDate,'yyyyMMdd') as int) DateKey,OrderDate FullDateAlternateKey,
format(OrderDate,'dddd','en-UK') EnglishDayNameOfWeek,
format(OrderDate,'dddd','nl-NL') DutchDayNameOfWeek,
month(OrderDate) MonthNumber,
format(OrderDate,'MMMM','en-UK') EnglishMonthName,
format(OrderDate,'MMMM','nl-NL') DutchMonthName,
```

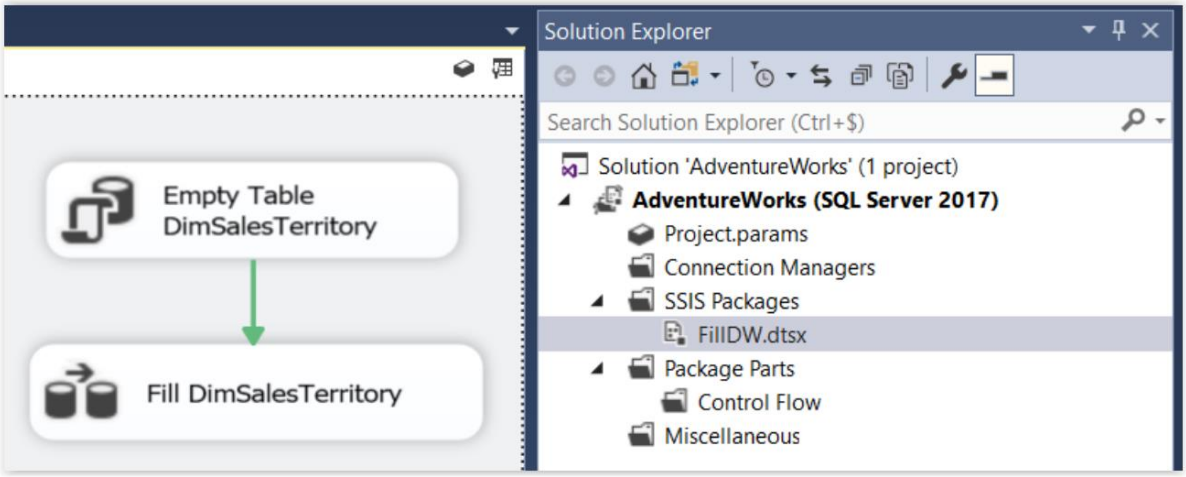
```
datepart(q,OrderDate) CalenderQuarter,datepart(YYYY,OrderDate) CalenderYear
from Sales.SalesOrderHeader
where OnlineOrderFlag = 1
```

Extract and load data

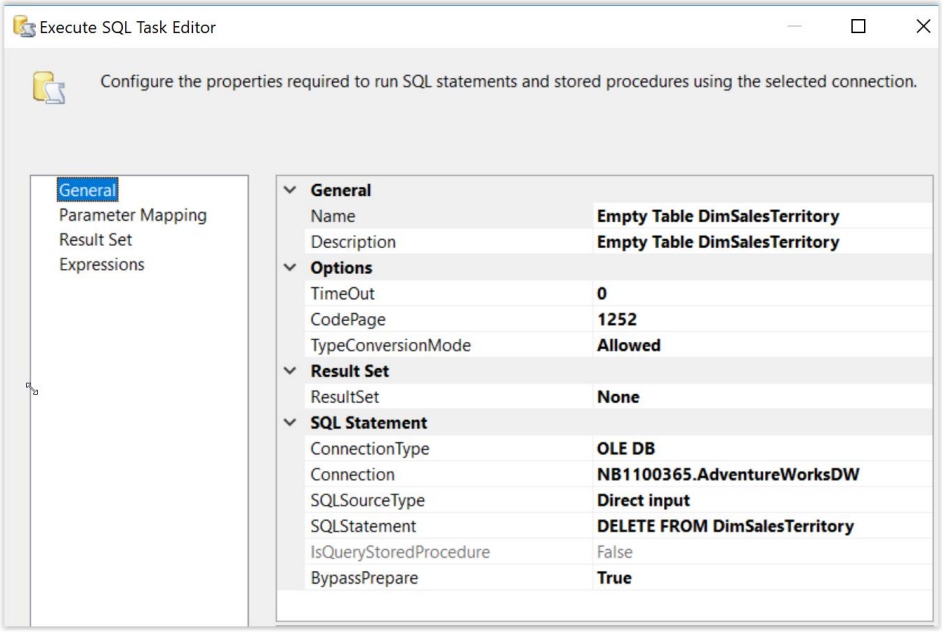
Fill dimensions

In the Solution Explorer rename package.dtsx to FillDW.dtsx

In the Control Flow tab drag and drop a "Execute SQL Task" and a "Data flow" task. Connect them and rename them as follows:

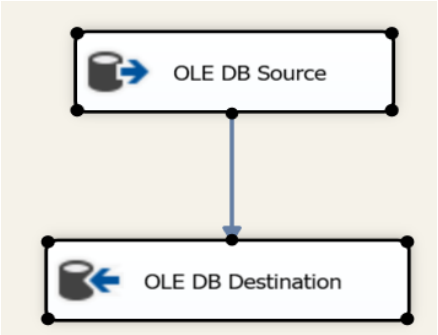


We want to delete all records from the table DimSalesTerritory before each transfer. Configure the " Empty table DimSalesTerritory" task:

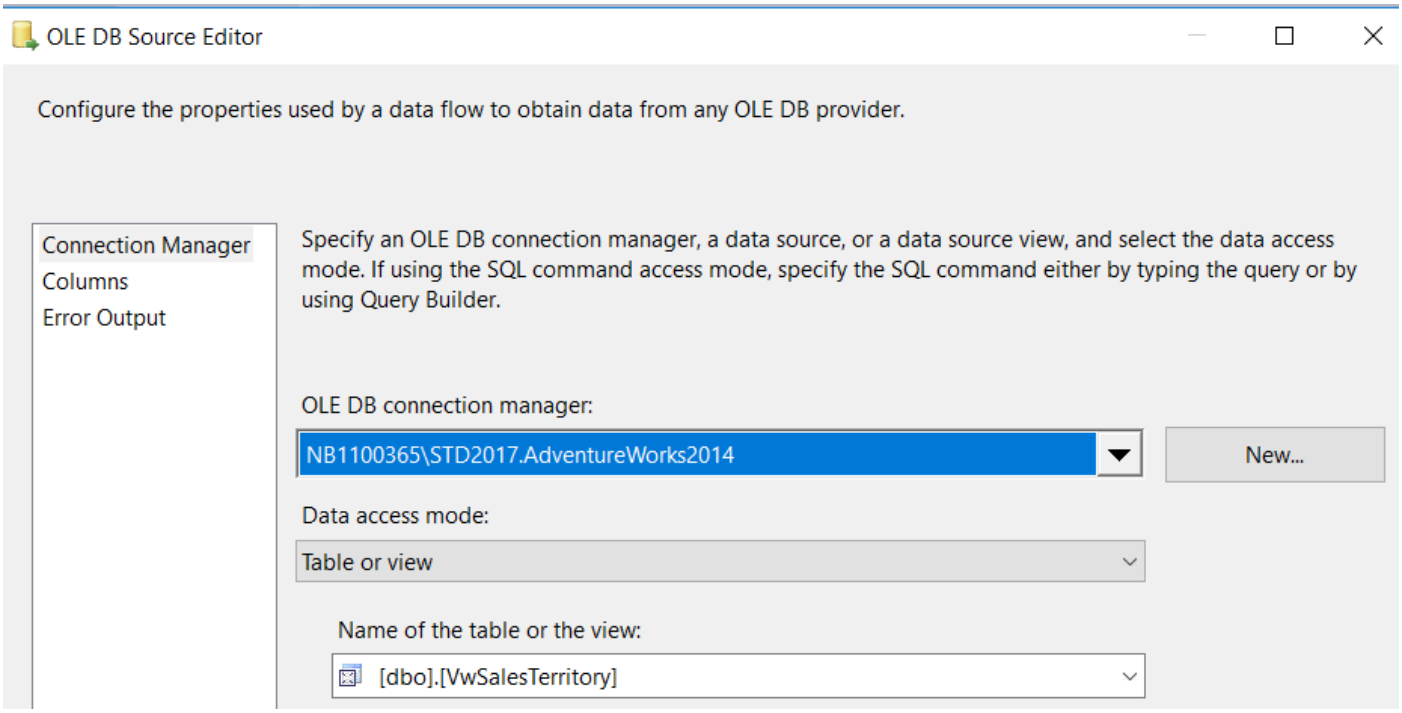


When double clicking the "Fill DimSalesTerritory" task the "Dataflow" tab is selected. Add and connect two data flow tasks: a source assistant and a destination assistant. In the source assistant make a connection to AdventureWorks2014 and in the destination assistant to AdventureWorksDW. Select the view

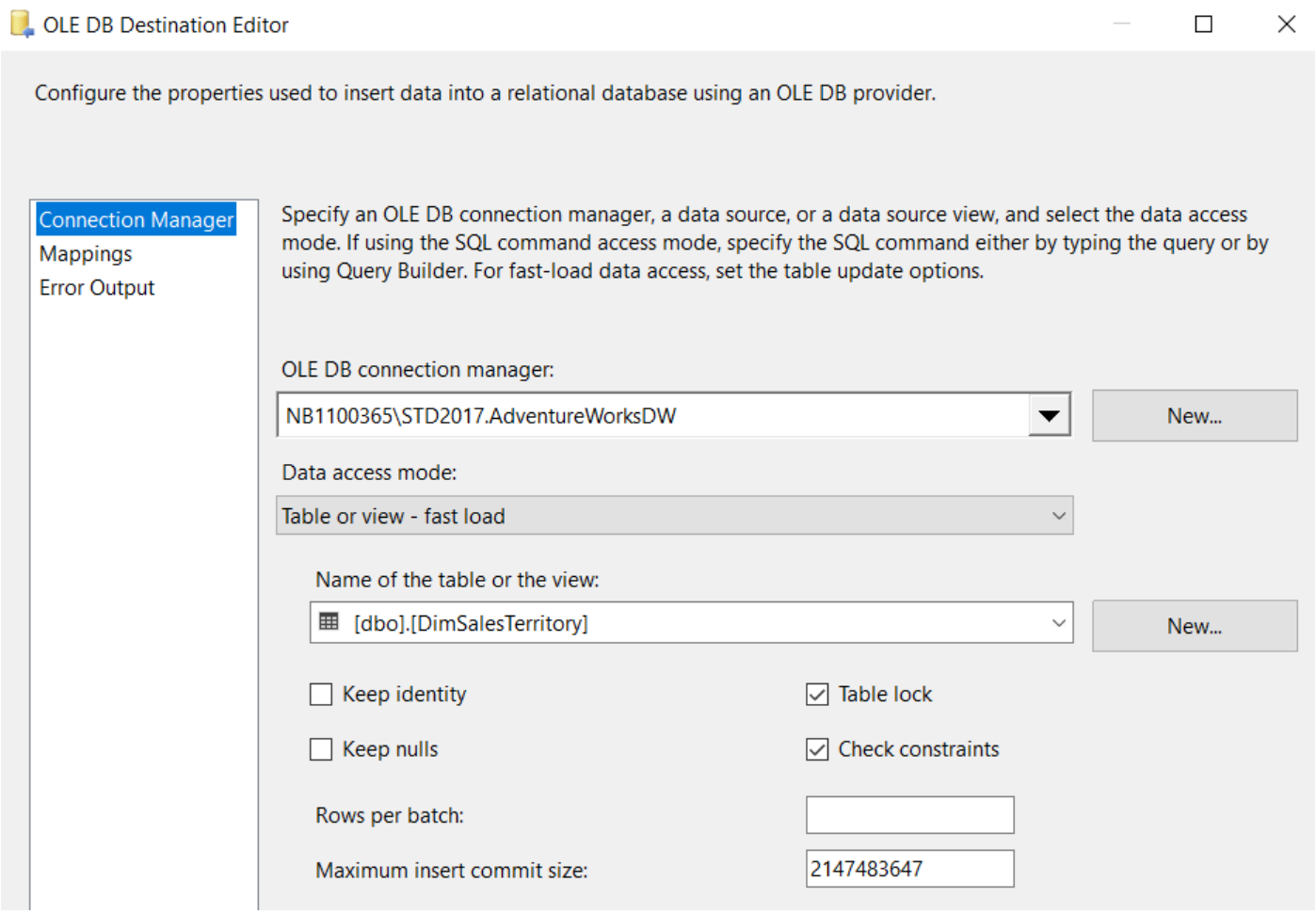
VwSalesTerritory in the source and the table DimSalesTerritory in the destination. Click and check "Mappings" in the destination.



Configure the tasks as follows:



Also check the Columns.



Check the Mappings to see if the corresponding columns are connected.

Go back to the Control Flow tab and click “Start”. Check the contents of the table “DimSalesTerritory”.

Now do the same for the dimension DimDate. Click start. You will get Unicode errors for the fields EnglishDayNameOfWeek, DutchDayNameOfWeek, etc.. This is due to the fact that the data types between the source data (from the view VwDimDate) and the destination data (the table DimDate in the datawarehouse) do not correspond. Indeed, EnglishDayNameOfWeek, DutchDayNameOfWeek, etc. are the result of the FORMAT function which returns a nvarchar (see https://docs.microsoft.com/en-us/sql/t-sql/functions/format-transact-sql?view=sql-server-2017). The corresponding fields in the datawarehouse are of type varchar. We choose to change the datatype in the view by doing a cast. Right click on the view VwDimDate and click "Script View as / Alter to / New query editor Windows". Change the view as follows:

```
select distinct cast(format(OrderDate,'yyyyMMdd') as int) DateKey,OrderDate
FullDateAlternateKey,
cast(format(OrderDate,'dddd','en-UK') as varchar) EnglishDayNameOfWeek,
cast(format(OrderDate,'dddd','nl-NL') as varchar) DutchDayNameOfWeek,
month(OrderDate) MonthNumber,
cast(format(OrderDate,'MMMM','en-UK') as varchar) EnglishMonthName,
cast(format(OrderDate,'MMMM','nl-NL') as varchar) DutchMonthName,
datepart(q,OrderDate) CalenderQuarter,datepart(YYYY,OrderDate) CalenderYear
from Sales.SalesOrderHeader
where OnlineOrderFlag = 1
```

Unfortunately, SSDT still remembers the "old" data types. Right click on OLE DB Source and choose Show Advanced Editor, go to the tab Input and Output Properties, expand OLE DB Source Output and Output Columns and look for EnglishDayNameOfWeek, DutchDayNameOfWeek, etc... Change the data type from Unicode String to String. Run the package and check the results.

Slowly changing dimension DimProduct

We now want to load the product data as a slowly changing dimension.

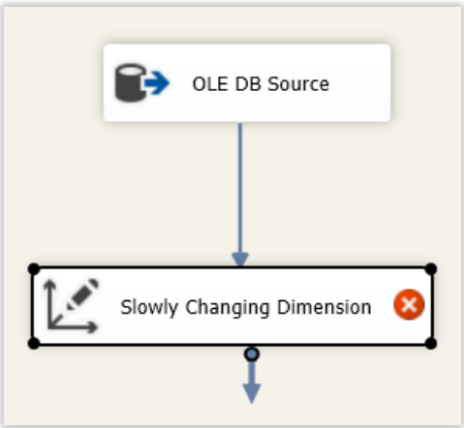
Create, in the datawarehouse, a table Product as follows:

Column Name	Data Type	Allow Nulls
ProductKey	int	☐
ProductID	int	☐
Name	nvarchar(50)	☑
Color	nvarchar(15)	☑
ListPrice	money	☑
Size	nvarchar(50)	☑
Weight	decimal(8, 2)	☑
Start	date	☑
[End]	date	☑

Specify ProductKey as an identity value with seed = 1 and increment = 1.

Add a task FillDimProduct Data Flow task. Don't add a delete task since we want to build product history!.

Double-click Fill DimProduct and add (in the Data Flow tab) an OLE DB Source and a Slowly Changing Dimension task:



Specify the Product.Product table from the AdventureWorks2014 database as the OLE DB Source.

Now follow the wizard for the Slowly Changing Dimension task.

Slowly Changing Dimension Wizard

Select a Dimension Table and Keys

Select a dimension table to load and map columns in the transformation input to columns in the dimension table.

Connection manager:

NB1100365.AdventureWorksDW | New...

Table or view:

[dbo].[DimProduct]

Input Columns	Dimension Columns	Key Type
Color	Color	Not a key column
	Current	
	End	
ListPrice	ListPrice	Not a key column
Name	Name	Not a key column
ProductID	ProductID	Business key
	ProductKey	
Size	Size	Not a key column
	Start	
Weight	Weight	Not a key column

Help | < Back | Next > | Finish >>| | Cancel

Slowly Changing Dimension Wizard

Slowly Changing Dimension Columns

Manage the changes to column data in your slowly changing dimensions by setting the change type for dimension columns.

Fixed Attribute

Select this type when the value in a column should not change. Changes are treated as errors.

Changing Attribute

Select this type when changed values should overwrite existing values. This is a Type 1 change.

Historical Attribute

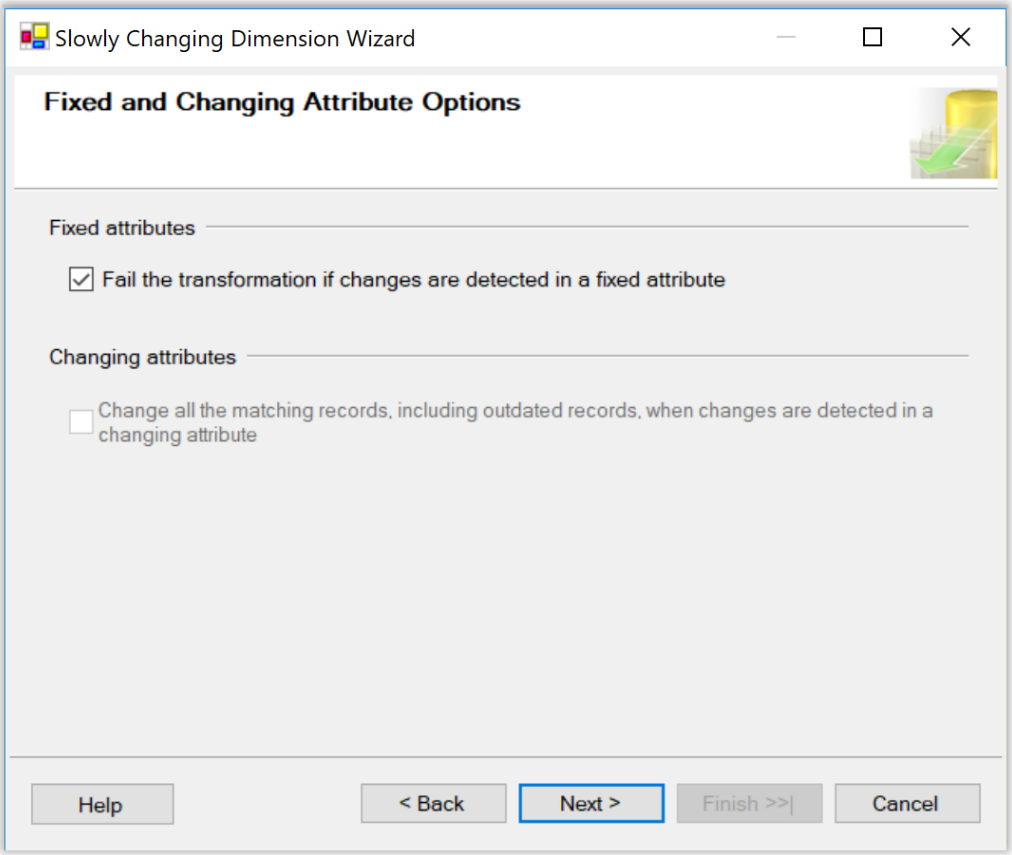
Select this type when changes in column values are saved in new records. Previous values are saved in records marked as outdated. This is a Type 2 change.

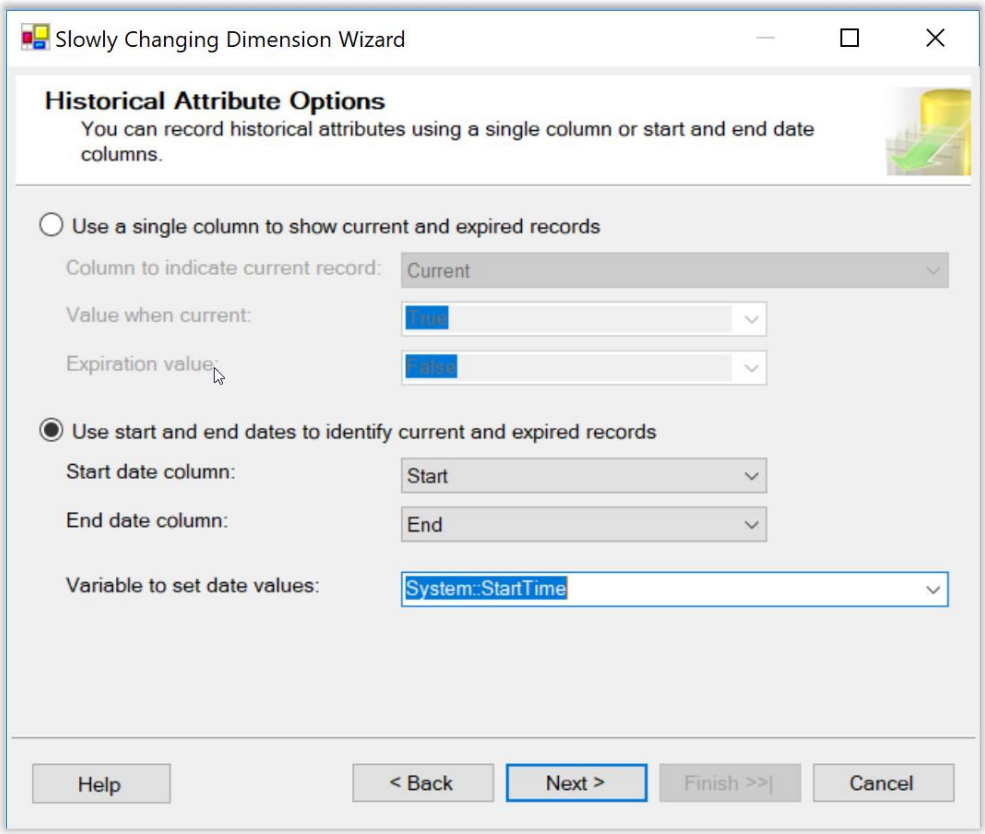
Select a change type for slowly changing dimension columns:

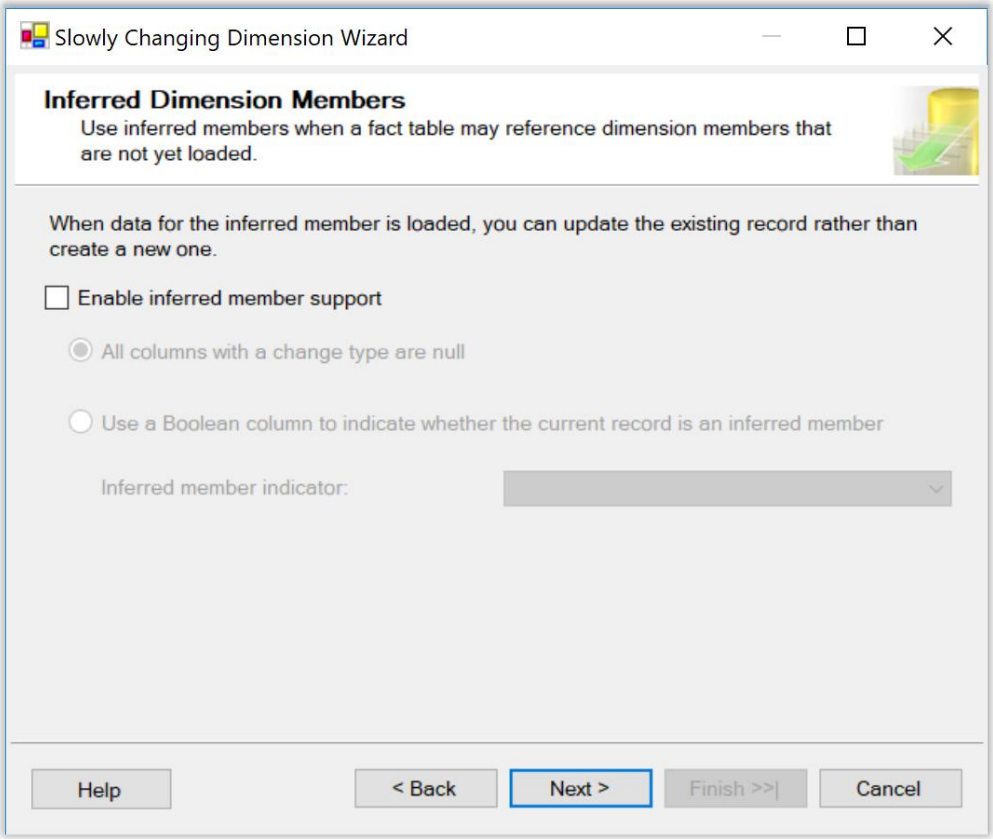
Dimension Columns	Change Type
Color	Historical attribute
ListPrice	Historical attribute
Name	Fixed attribute
Size	Historical attribute
Weight	Historical attribute

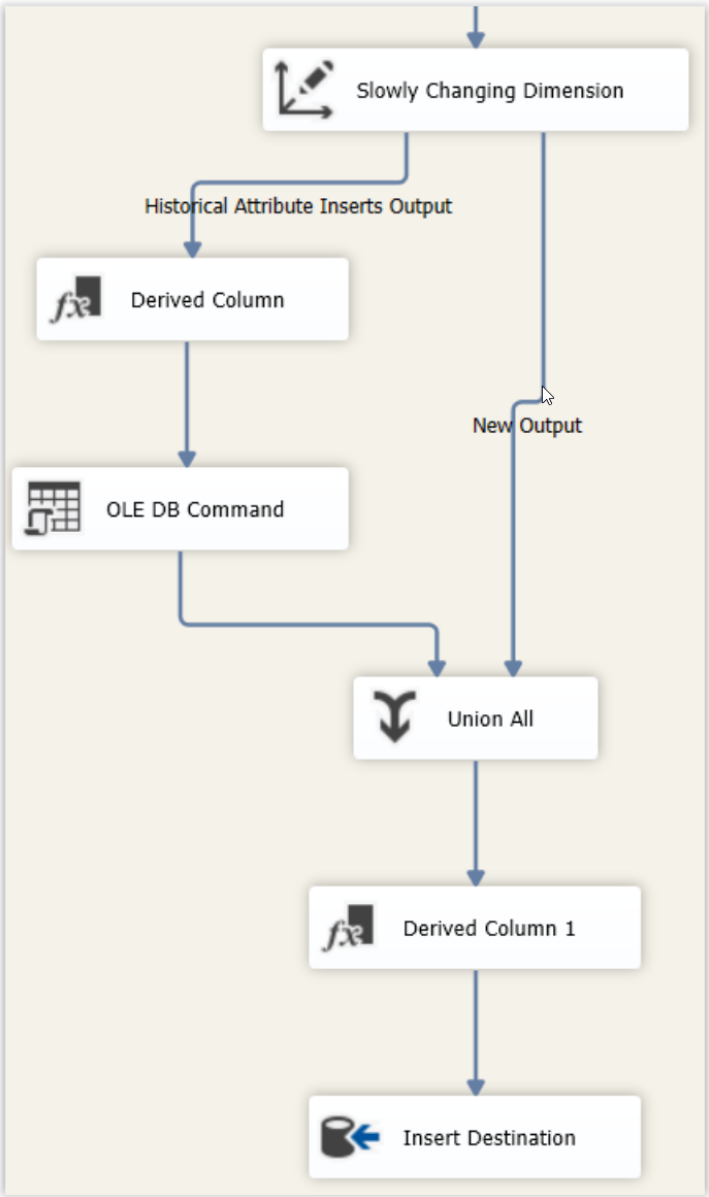
Remove

Help | < Back | Next > | Finish >>| | Cancel


Slowly Changing Dimension Wizard
Fixed and Changing Attribute Options
Fixed attributes
Fail the transformation if changes are detected in a fixed attribute
Changing attributes
Change all the matching records, including outdated records, when changes are detected in a changing attribute
Help
< Back
Next >
Finish >>|
Cancel


Slowly Changing Dimension Wizard
Historical Attribute Options
You can record historical attributes using a single column or start and end date columns.
Use a single column to show current and expired records
Column to indicate current record:
Current
Value when current:
True
Expiration value:
False
Use start and end dates to identify current and expired records
Start date column:
Start
End date column:
End
Variable to set date values:
System::StartTime
Help
< Back
Next >
Finish >>|
Cancel


Slowly Changing Dimension Wizard
Inferred Dimension Members
Use inferred members when a fact table may reference dimension members that are not yet loaded.
When data for the inferred member is loaded, you can update the existing record rather than create a new one.
Enable inferred member support
All columns with a change type are null
Use a Boolean column to indicate whether the current record is an inferred member
Inferred member indicator:
Help
< Back
Next >
Finish >>|
Cancel



Check the result of running the whole project until now:

- DimProduct is filled but start = today. From a business point of view this is essentially wrong: current product data is valid since the start of the company (since we have no history yet). So execute:

```
update dimproduct set start =
(select min(orderdate) from AdventureWorks2014.Sales.SalesOrderHeader);
```

Check what happens is e.g. the color of a product is updated and you rerun the package:

1. On the OLTP database: `update production.product set color = 'Blue' where productid = 776;`
2. Rerun the package
3. On the datawarehouse: `select * from dimproduct where productid = 776;`
   ➔ There are now two lines of this product.

Fil facts

First check if field SalesOrderDetailID from the table AdventureWorks2014.Sales.SalesOrderDetail is a candidate primary key (in that table it forms a primary key together with SalesOrderId):

```
select
(select count(SalesOrderDetailID)
from AdventureWorks2014.Sales.SalesOrderDetail)
-
(select count(distinct SalesOrderDetailID)
from AdventureWorks2014.Sales.SalesOrderDetail);
```

➔ Returns 0 → OK.

Create table FactSales:

Column Name	Data Type	Allow Nulls
SalesOrderLineNumber	int	☐
ProductKey	int	☐
SalesTerritoryKey	int	☐
OrderDateKey	int	☐
OrderQuantity	smallint	☐
UnitPrice	money	☐
ExtendedAmount	money	☐
		☐

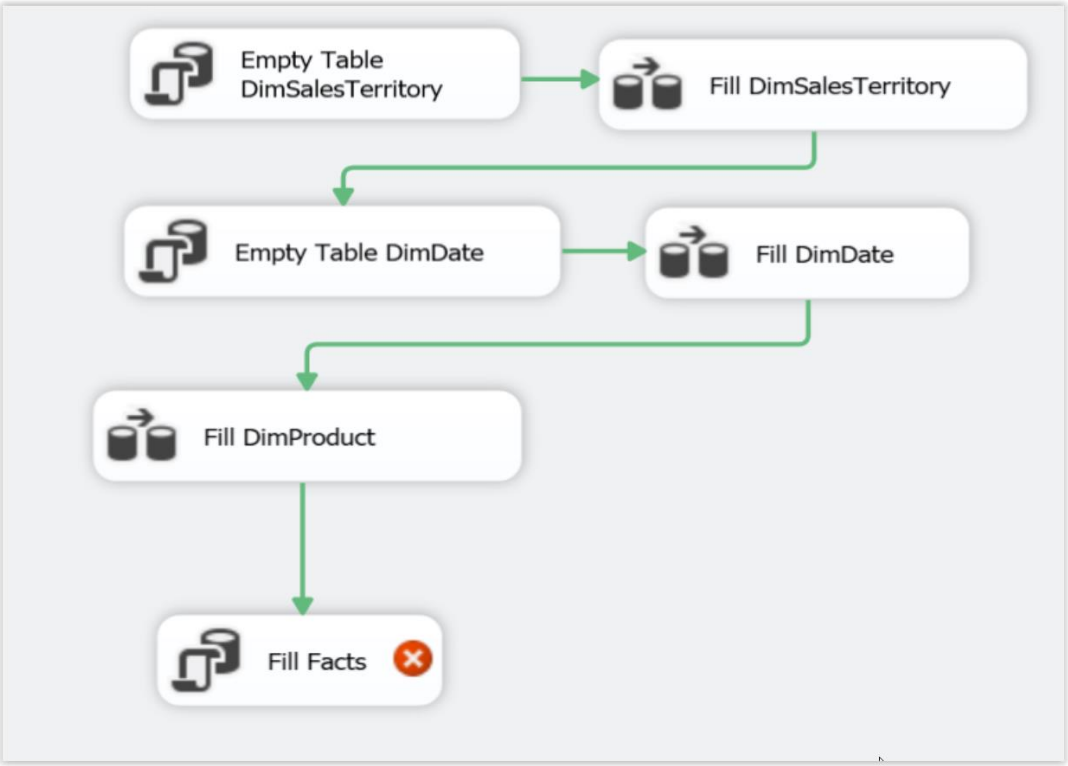
We can now fill the table FactSales repeatedly with the command:

```
insert into
factsales(SalesOrderLineNumber,ProductKey,SalesTerritoryKey,OrderDateKey,OrderQuantity,UnitPrice,Ext
endedAmount)
select d.SalesOrderDetailID,p.ProductKey,h.TerritoryID,cast(format(h.OrderDate,'yyyyMMdd') as
int),d.OrderQty,d.UnitPrice,
d.OrderQty * d.UnitPrice
from AdventureWorks2014.Sales.SalesOrderHeader h join
AdventureWorks2014.Sales.SalesOrderDetail d on h.SalesOrderID = d.SalesOrderID
join DimProduct p on d.ProductID = p.ProductID
where
/* Slowly Changing Dimension dimproduct */
h.OrderDate >= p.start and (p.[end] is null or h.orderdate < p.[end])
and /* increment, also make sure it runs from an empty factsales table */
d.SalesOrderDetailID > (select isnull(max(SalesOrderLineNumber),0) from factsales)
;
```

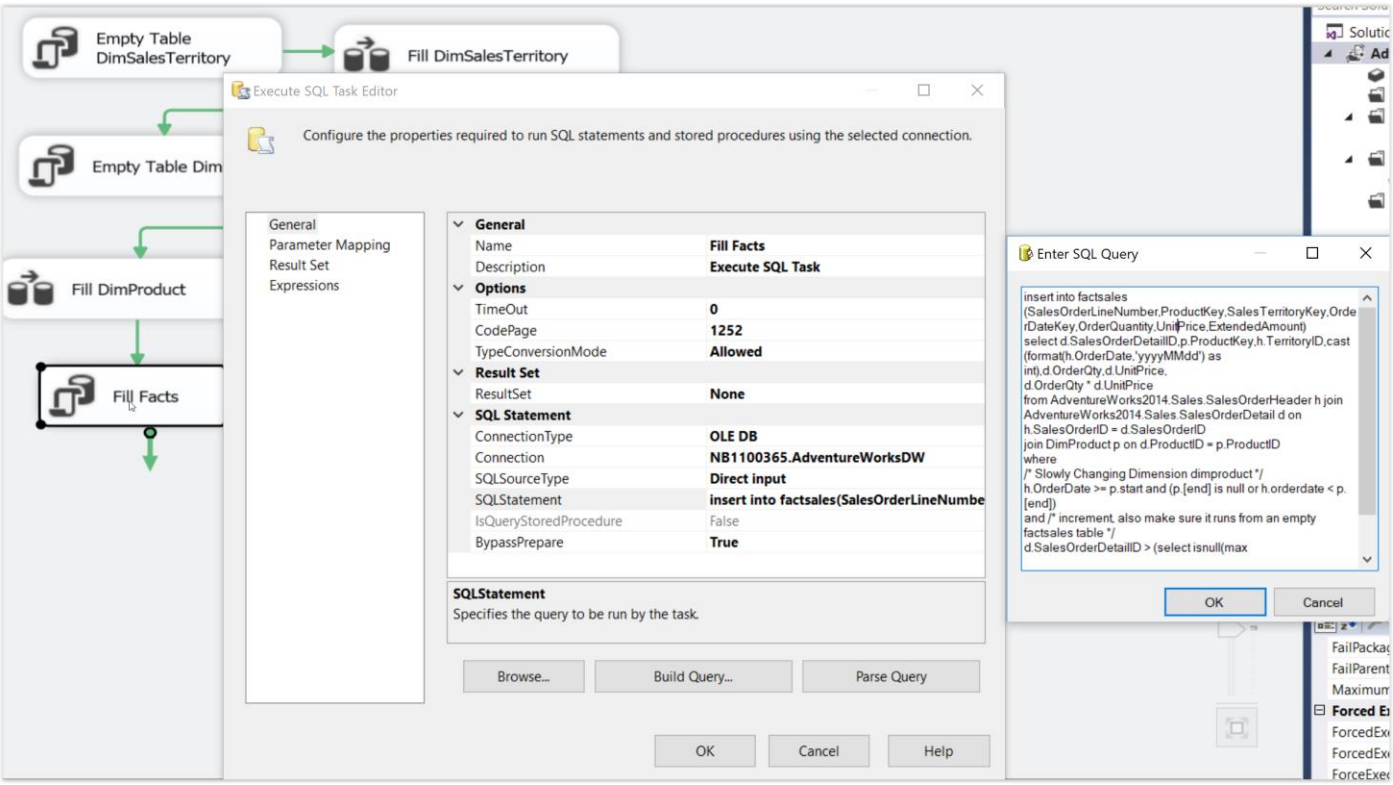
Remarks:

- ProductKey (and not ProductID, which is kept as a business key) is inserted as foreign key so we can also link to the correct Product information when making sales reports.
- The first where clause makes sure that we join with the correct dimproduct line so we can insert the correct ProductKey.
- Due to the second where clause we only add new lines to factsales in consecutive runs of this statement.

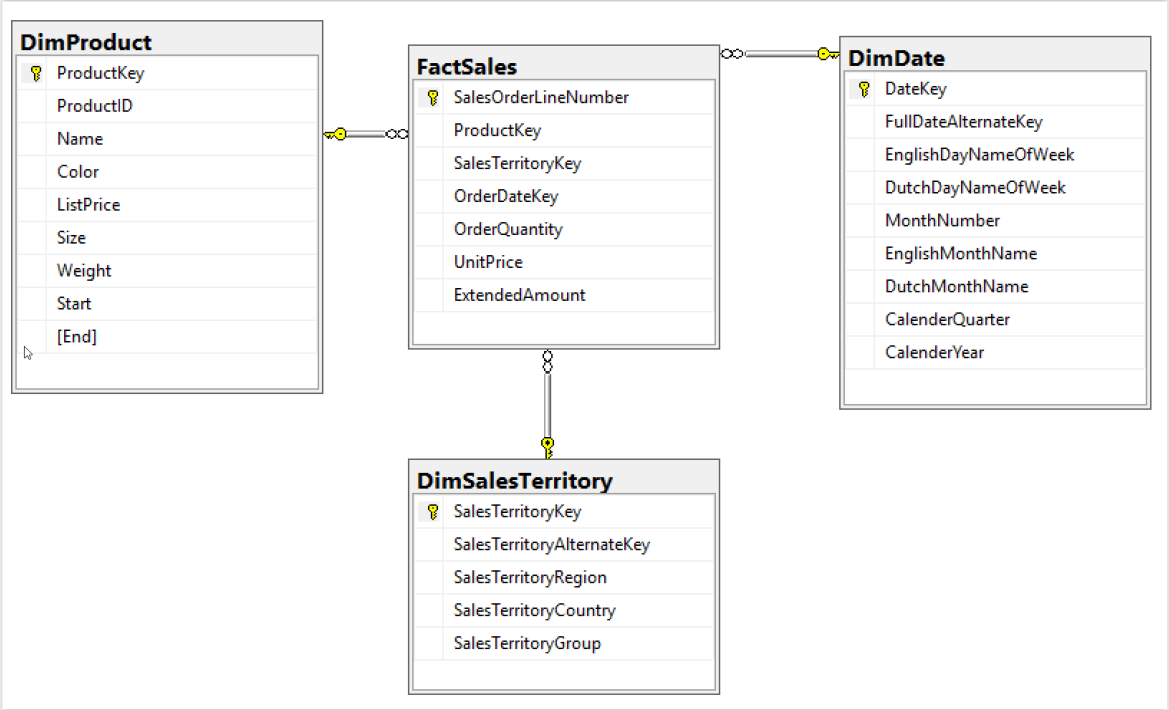
Add an Execute SQL Taks as the last task in your design:



Copy the above insert command in this task:



Finally create a database diagram for the datawarehouse and draw the foreign key constraints:

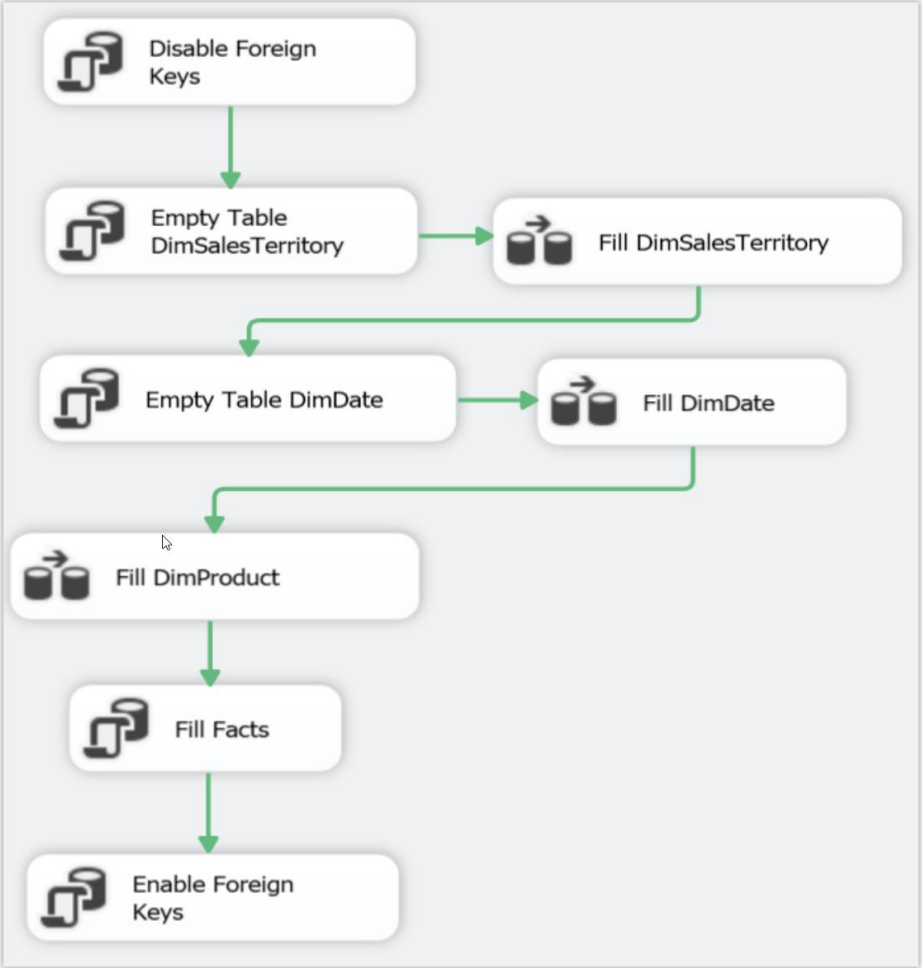


Now run the SSDT project as a whole and check the resulting tables. We get an error message because we are trying to delete DimDate and DimSalesTerritory while there are foreign key constraints between those tables and FactSales. One way to cope with this is to temporarily disable the constraints at the start of the fill operation and enable them again at the end. So add two Execute SQL tasks:

At the start: `ALTER TABLE FactSales NOCHECK CONSTRAINT ALL;`

At the end: `ALTER TABLE factsales WITH CHECK CHECK CONSTRAINT ALL;` (double CHECK!)

So the final schema looks like this:



As a final test we can add a sales line for an updated product in the operational system and check if the corresponding FactSales line in the datawarehouse will be linked to the correct ProductKey in DimProduct:

Copy the last line from [Sales].[SalesOrderHeader] (SalesOrderID is an identity column, so we have to specify all fields)

```
INSERT INTO [Sales].[SalesOrderHeader]

([RevisionNumber],[OrderDate],[DueDate],[ShipDate],[Status],[OnlineOrderFlag],[PurchaseOrderNumber]

,[AccountNumber],[CustomerID],[SalesPersonID],[TerritoryID],[BillToAddressID],[ShipToAddressID]
           ,[ShipMethodID],[CreditCardID],[CreditCardApprovalCode],[CurrencyRateID],[SubTotal]
           ,[TaxAmt],[Freight],[Comment],[rowguid],[ModifiedDate])
select s.RevisionNumber,'2019-01-23','2019-02-23','2019-02-23',s.Status,
s.OnlineOrderFlag,s.PurchaseOrderNumber,s.AccountNumber,s.CustomerID,
s.SalesPersonID,s.TerritoryID,s.BillToAddressID,s.ShipToAddressID,
s.ShipMethodID,s.CreditCardID,s.CreditCardApprovalCode,s.CurrencyRateID,
s.SubTotal,s.TaxAmt,s.Freight,s.Comment,newid(),getdate()
from sales.salesorderheader s
where salesorderid = (select max(salesorderid) from sales.salesorderheader);
```

Check for the key value to use in [Sales].[SalesOrderDetail]:

```
select max(salesorderid) from sales.salesorderheader; --> 75125
```

Add a line to the above created SalesOrderHeader:

```
INSERT INTO [Sales].[SalesOrderDetail]([SalesOrderID],[CarrierTrackingNumber],[OrderQty],[ProductID]
           ,[SpecialOfferID],[UnitPrice],[UnitPriceDiscount],[rowguid],[ModifiedDate])
     VALUES (75125,null,4,776,1,10,0,newid(),getdate());
```

Now rerun the package and check factsales:

```
select * from dimproduct where productid = 776;
```

	ProductKey	ProductID	Name	Color	ListPrice	Size	Weight	Start	End
1	281	776	Mountain-100 Black, 42	Black	3374,99	42	20.77	2011-05-31	2019-01-23
2	506	776	Mountain-100 Black, 42	Blue	3374,99	42	20.77	2019-01-23	NULL

506 is the correct ProductKey.

```
Select top 1 * from factsales order by salesorderlinenumber desc;
```

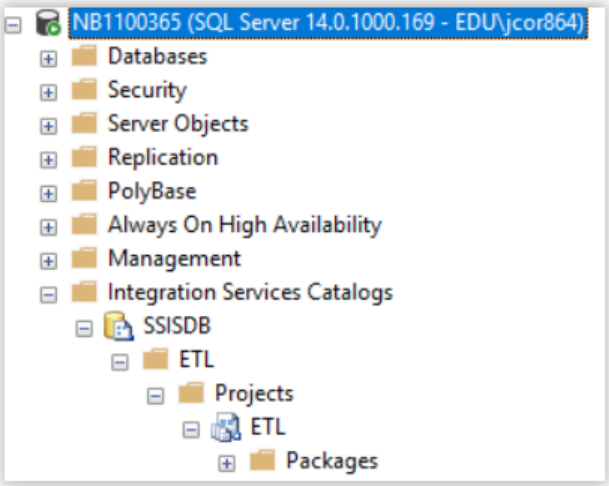
	SalesOrderLineNumber	ProductKey	SalesTerritoryKey	OrderDateKey	OrderQuantity	UnitPrice	ExtendedAmount
1	121320	506	6	20190123	4	10,00	40,00

➔ Correct!

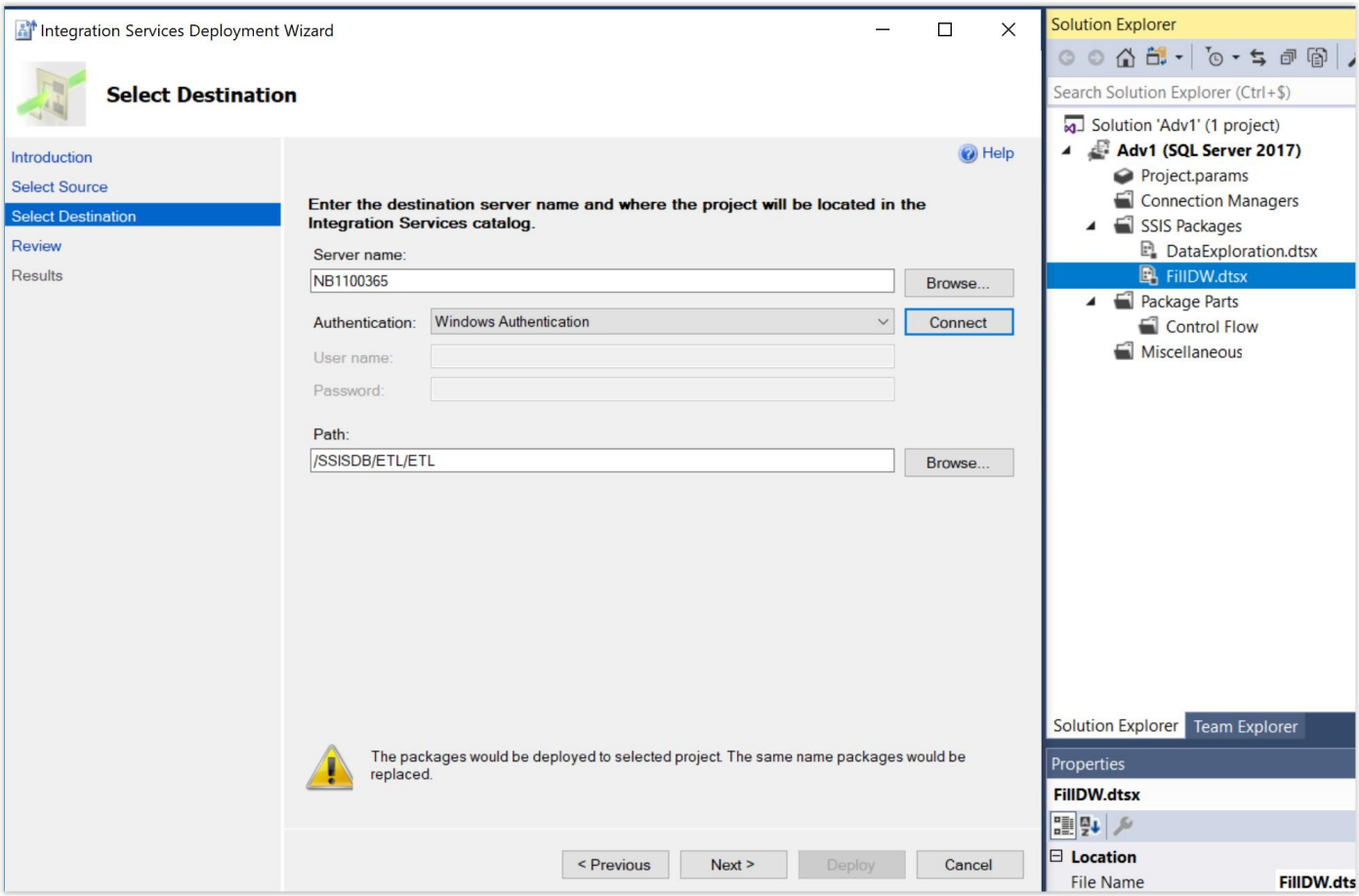
Deploy and schedule the package

In Microsoft SQL Server Enterprise Edition you can run your package from within SQL Server Management Studio and also schedule the package, e.g. every night at 1 am.

In SQL Server Management Studio create, under Integration Services Catalogs a hierarchy of a Catalog (SSISDB), a folder (ETL) and a project (ETL):

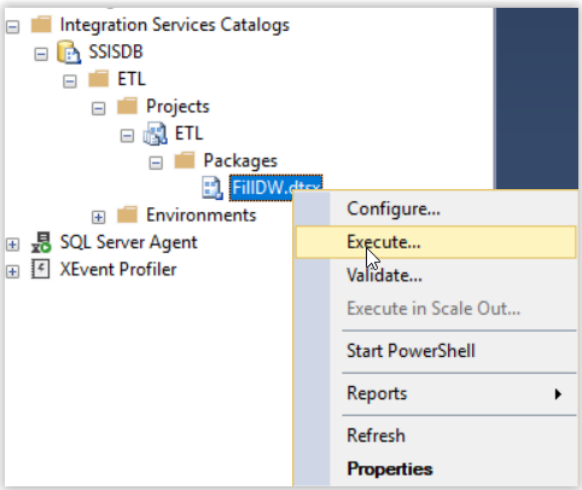


No go back to SSDT. Right-click on the package FillDW.dtsx and choose Deploy. Follow the wizard that pops up:



Fill in you server name. In a production environment this normally not your own development machine.

Click Connect in the form above and Next. Enter the path you have just created in Management Studio. In the last form click Deploy. If everything went well you will now see you deployed package in SSMS:



Right-click on it and Execute.

Now, still in SSMS, go to “SQL Server Agent” (if it shows a red icon, right-click to start) and create a new job. Choose your SSIS package as the only step in the job and create a schedule. The package will now run automatically according to the schedule.